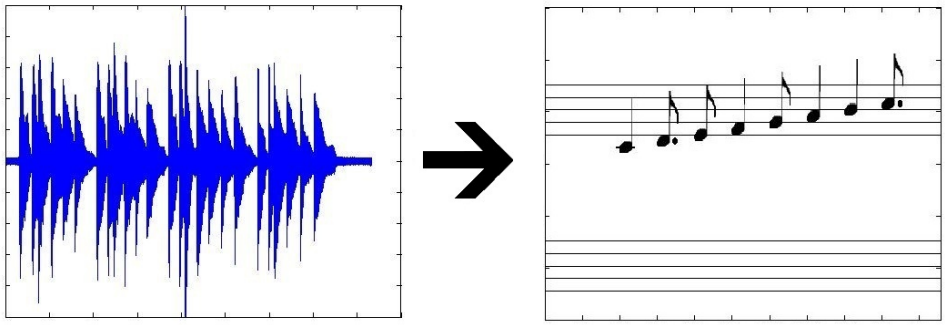
Sound Analysis using STFT spectroscopy

by

Tobias Maas

Thesis
for the Degree of
Bachelor of Science
in Industrial Mathematics

University of Bremen



1. Evaluator: Dr. Stefan Schiffler, University of Bremen
2. Evaluator: Dr. Dennis Trede, University of Bremen

July 22, 2011

Contents

List of Figures

Chapter 1

Introduction

1.1 Outline

This bachelor thesis focuses on the concept of extracting information from sound, more precisely extracting the played pitches of given instruments at all times. For this to work, one has to know how to physically and mathematically describe the nature of sound.
This is discussed in the first part of this thesis, where the concept of the Fourier transformation for frequency analysis is presented. For changing frequencies, the Fourier transformation of the whole function is not sufficient, so the transformation has to be specialized to the Short Time Fourier Transformation (STFT) to get a time dependent analysis. The second part engages the possibilities of comparing incoming sounds with those who are already analysed to find corresponding pitches and instruments at each time. Mainly the simple L_2-comparison, the sparsity algorithm and the elastic-net algorithm are presented and discussed to find a sufficient algorithm for a given problem. With all these theoretical aspects clarified, the third part presents an actual implementation using a dictionary of sounds and the sparsity algorithm for the comparison. The whole algorithm can be found in the appendum. The last part concludes the thesis, giving an outlook to the future of this topic.
Extracting informations out of sound is engaged by several researchers. The leading company on this market is Celemony. Their Melodyne editor can seperate and modify the notes of easy songs, so that wrongly played pitches can be corrected or the tune of the melody can be changed. More informations about the Melodyne editor can be found at [9]. The main motivation for this thesis was to create a program that could decode any arbitrary sound file and output the score of its melody.

1.2 The physics of music

As Klapuri and Davy define it (cf.[3]), a basic theoretical tone is a sound wave oscillating with one base frequency between 20 Hz and 20 kHz, originating from the cause of the tone. These waves are called sound waves, because the eardrum inside the human ear vibrates in resonance with the incoming sound, transfering signals to the brain. Within this spectrum the brain reinterprets the signal as sound. The amplitude of the oscillating wave is proportional to the oscillation of the eardrum, resulting in a proportional volume.

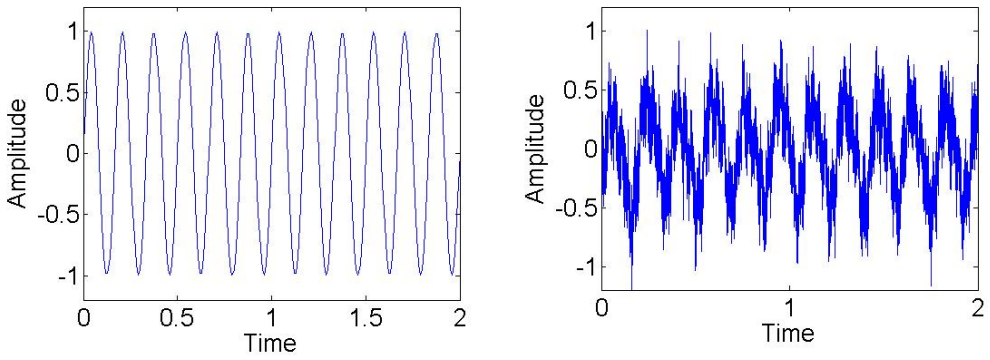


Figure 1.1: Basic tone (left), Real tone (right)

Basic tones differ from real tones in many ways.
Real instruments create their own characteristics of a tone by adding a complex structure of overtones, more precisely adding natural multiples of its base frequency. Some instruments, e.g. a trumpet, additionally create non-constant frequencies because of their mouth piece or vibrations inside their bodies. These effects create the hearing experience as we know it, but are physically and mathematically harder to describe.
When speaking about music, one has to notice that sound waves do not have constant frequencies or amplitudes. In addition to that, several pitches and instruments are played at once, resulting in superposed uninterrupting sound waves. Seperating this mass of frequencies as played pitches is the topic of the next chapter.

Chapter 2

Theoretical Background

2.1 Continous approaches

2.1.1 The Fourier transformation

In 1822 Jean Baptiste Joseph Fourier laid the foundation for analysing frequencies(cf.[4]). The so called Fourier transform transfers a function into the corresponding frequency space to easily extract informations about the frequency spectrum. The easiest way to do so is by using the complex

$$e^{ikx} = \cos(kx) + i\sin(kx), \quad |e^{ikx}| = |\cos(kx) + i\sin(kx)| = 1$$

with the reversion

$$\cos(kx) = \frac{1}{2}\left(e^{ikx} + e^{-ikx}\right), \quad \sin(kx) = \frac{1}{2i}\left(e^{ikx} - e^{-ikx}\right).$$

This attribute can be used to obtain the absolute value of the amplitude of a given sinus or cosinus oscillation. As mentioned before, musical compositions are created out of overlapping oscillations, so they can be described as a sum of different sinuses and cosinuses. The absolute value ignores the phase difference between the two oscillations, so this sum can be simplified even further by only using sinuses. Another effect of their oscillating character is that the compositions we want to analyse are infinitely differentiable and of finite length. It follows that every function has its support inside a compact cube, so that any supremum or infimum is reached inside. Another benefit of finite support is the fact, that infinitely differentiable functions are automatically in L_1 and L_2.
In this thesis the support is normalised to $[0, 2\pi]$ to not have to use scalar transformations of the period of e^{ikx}, but all results can be transferred to any finite interval. With the demand of f being infinitely differentiable, the following lemma holds true.

Lemma 2.1.1. *Let f be in $C^\infty[0,2\pi]$. Then $\frac{1}{2\pi} f e^{-ikx}$ is in $C^\infty[0,2\pi] \subset L_1[0,2\pi]$ aswell.*

Proof. Let f be in $C^\infty[0,2\pi]$. Because e^{-ikx} is also in $C^\infty 0,2\pi]$ and the multiplication of two infinitely differentiable functions is in $C^\infty[0,2\pi]$ aswell, it follows that $\frac{1}{2\pi} f e^{-ikx}$ is in $C^\infty[0,2\pi]$. For $L_1[0,2\pi]$, examine the following equation:

$$\int_0^{2\pi} |f(x)e^{-ikx}|dx = \int_0^{2\pi} |f(x)|dx \leq 2\pi \max_{x\in[0,2\pi]} |f(x)| \tag{2.1}$$

$\Rightarrow f e^{-ikx} \in L_1[0,2\pi]$. □

For further details on this topic, see [1].
Having this clarified, we can define the Fourier transform as it can be found in ([2],p.103) .

Theorem 2.1.2. *Let u be in $L_1[0,2\pi]$ and $\xi \in \mathbb{R}$. Then*

$$(Fu)(\xi) = \hat{u}(\xi) := \int_0^{2\pi} u(x)e^{-ix\xi}dx \tag{2.2}$$

is called the Fourier transformation of u. The function $F : u \mapsto \hat{u}$ is called Fourier transform. F is well defined.

Proof. It has to be shown, that this integral is well defined.

$$\int_0^{2\pi} |u(x)e^{-ix\xi}|dx = \int_0^{2\pi} |u(x)||e^{-ix\xi}|dx = \int_0^{2\pi} |u(x)|dx \leq ||u||_{L_1} < \infty \tag{2.3}$$

Because u is in $L_1[0,2\pi]$, the integral is well defined. □

This transform helps to identify the frequencies of a given function. Suppose $f(x) = 2e^{i3x} + 4e^{i5x}$. With the previous lemma we can compute the Fourier transformation

$$\begin{aligned}(Fu)(\xi) &= \int_0^{2\pi} e^{-i\xi x} f(x)dx \\ &= \int_0^{2\pi} e^{-i\xi x} 2e^{i3x}dx + \int_0^{2\pi} e^{-i\xi x} 4e^{i5x}dx \\ &= \int_0^{2\pi} 2e^{i(3-\xi)x}dx + \int_0^{2\pi} 4e^{i(5-\xi)x}dx\end{aligned}$$

as

$$\begin{cases} 2 \cdot 2\pi, & \text{if } \xi = 3 \\ 4 \cdot 2\pi, & \text{if } \xi = 5 \\ \frac{2i}{3-\xi}(1 - e^{i(3-\xi)2\pi}) + \frac{4i}{5-\xi}(1 - e^{i(5-\xi)2\pi}), & \text{else} \end{cases}$$

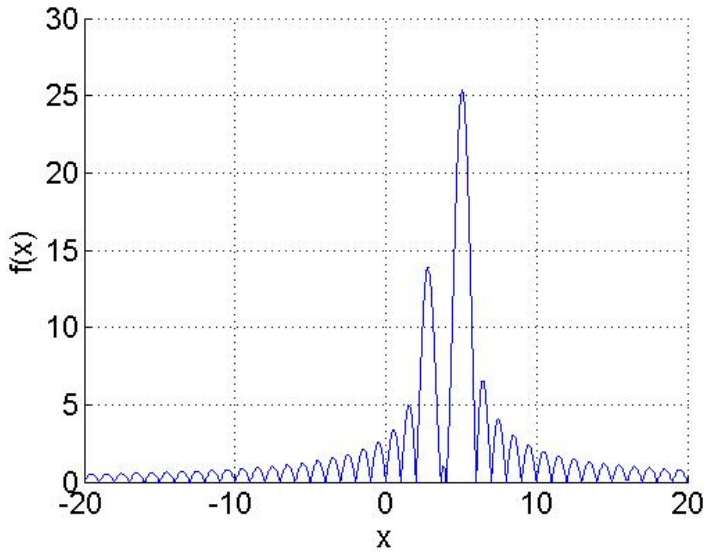


Figure 2.1: Fourier transformation of $f(x) = 2e^{i3x} + 4e^{i5x}$

This compution returns the amplitudes of the given frequencies, peaking in the real frequencies. If we can now show that the Fourier transform is bijective, we can use these amplitudes in the frequency space to get information about the corresponding composition. However, this is not possible in $L_1[0, 2\pi]$. We have to limit our space to the so called **Schwartz space**, as defined by the following definition taken out of ([2],p.106).

Definition 2.1.3. *The space*

$$S([0, 2\pi]) = \{u \in C^\infty | \forall \alpha, \beta \in \mathbb{N} : C_{\alpha,\beta} = \sup_{x \in [0,2\pi]} |x^\alpha \frac{\partial^\beta}{\partial x^\beta} u(x)| < \infty\} \quad (2.4)$$

is called ***Schwartz space*** *and elements $u \in S([0, 2\pi])$ are called Schwartz functions.*

One can say that elements of $S([0, 2\pi])$ are faster vanishing than any polynomial can converge towards ∞. This space is stronly linked with the Fourier

transform, because the requirement towards functions in $S([0,2\pi])$ have an interesting attribute in the Fourier transform.

Lemma 2.1.4. *Let u be in $S([0,2\pi])$ and $\alpha \in N$. Then*

$$F(\frac{\partial^\alpha u}{\partial x^\alpha}) = i^{|\alpha|} x^\alpha F(u) \tag{2.5}$$

$$F(x^\alpha u) = i^{|\alpha|} \frac{\partial^\alpha}{\partial x^\alpha} F(u) \tag{2.6}$$

Proof. A proof can be found in ([2], pp.107-108). □

With this lemma, we can show the bijectivity of F on $S([0,2\pi])$ as it can be found in ([2], p.111).

Theorem 2.1.5. *The Fourier transform $F : S([0,2\pi]) \to S([0,2\pi])$ is bijective . For $u \in S([0,2\pi])$ the inversion is given by*

$$(F^{-1}Fu)(x) = \check{\hat{u}} = \int_0^{2\pi} \hat{u}(\xi) e^{ix\xi} d\xi = u(x) \tag{2.7}$$

Proof. Let u and φ both be in $S([0,2\pi])$. Then the convultion of $\hat{\hat{u}}$ and φ is equal to the negative convultion of $\hat{\hat{\varphi}}$ and u. This holds true because of elementary convultion attributes.

$$\begin{aligned}(\hat{\hat{u}} * \varphi)(x) &= \int_0^{2\pi} \hat{\hat{u}}(y)\varphi(x-y)dy = \int_0^{2\pi} \hat{u}(y) e^{iky} \hat{\varphi}(-y) dy \\ &= \int_0^{2\pi} u(y) \hat{\hat{\varphi}}(-x-y) dy = (u * \hat{\hat{\varphi}})(-x)\end{aligned}$$

For φ as an rescaled gaussian function $\varphi(x) = \epsilon^{-d} e^{-\frac{|x|^2}{2\epsilon^2}}$ the equation $\hat{\hat{\varphi}} = \varphi$ follows, because the gaussian function is an eigenfunction of the Fourier transform. With $\epsilon \to 0$ the convultion converges to

$$\hat{\hat{u}} * \varphi(x) \to \hat{\hat{u}}(x) \quad \text{and} \quad u * \varphi(-x) \to u(-x)$$

so in conclusion one can say that

$$\hat{\hat{u}}(x) = u(-x)$$

Because of the change of sign between the transform $e^{-ix\xi}$ and the inversion $e^{ix\xi}$, the inversion $\check{\hat{u}}$ can be written as $-\hat{\hat{u}}$, so it follows that

$$\check{\hat{u}} = u$$

□

If we can now show, that a continous and differentiable composition is inside $S([0, 2\pi])$, we can use the previous theorem to transfer it to the frequency space and back.

Theorem 2.1.6. $C^\infty[0, 2\pi] \subset S([0, 2\pi])$.

Proof. Suppose $f \in C^\infty[0, 2\pi]$. Fix $\alpha, \beta \in N$. Then $\frac{\partial^\beta}{\partial x^\beta} u(x)$ is in $C^\infty[0, 2\pi]$ aswell. Because of the compactness of $[0, 2\pi]$, f reaches its supremum inside the interval. So altogehter:

$$\begin{aligned}
&\sup_{x\in[0,2\pi]} |x^\alpha \frac{\partial^\beta}{\partial x^\beta} u(x)| \\
&\leq \sup_{x\in[0,2\pi]} |x^\alpha| \sup_{x\in[0,2\pi]} |\frac{\partial^\beta}{\partial x^\beta} u(x)| \\
&\leq |(2\pi)|^\alpha \sup_{x\in[0,2\pi]} |\frac{\partial^\beta}{\partial x^\beta} u(x)| \\
&< \infty
\end{aligned}$$

□

In conclusion, we can identify the frequencies of a given composition by transforming it into the frequency space. Because this transform is bijective, the frequencies can be used to compare it to a given spectrum. But if we try to analyse a real piece of music, the results are not satisfying. As noticed before, in real music the frequencies change over time, but this effect is completely neglected by using the fourier transformation on the whole function.

2.1.2 Short Time Fourier Transformation (STFT)

A possible solution for this problem is to lay focus on one area at a time. This can be done by multiplying the function with a fast vanishing window function in $C^\infty([0, 2\pi])$ that is centered around the focus point. As ([2], p. 135) formulate it, the Fourier theorem can be restated as

Theorem 2.1.7 (The short-time Fourier transformation)**.** *Let f and g be in $C^\infty[0, 2\pi] \subset S([0, 2\pi])$, $x_0 \in [0, 2\pi]$ and $\xi \in \mathbb{R}$. Then*

$$(F_g u)(\xi, x_0) = \int_0^{2\pi} f(x) g(x - x_0) e^{-i\xi x} d\xi \tag{2.8}$$

is called the Short Time Fourier Transform of f in respect to x_0. F is well defined.

Proof.

$$\begin{aligned}
&\int_0^{2\pi} |f(x)g(x-x_0)(x)e^{-i\xi x}|dx \\
&= \int_0^{2\pi} |f(x)g(x-x_0)|dx \\
&\leq \max_{x\in[0,2\pi]} |f(x)| \int_0^{2\pi} |g(x-x_0)|dx \\
&\leq 2\pi \max_{x\in[0,2\pi]} |f(x)| \max_{x\in[0,2\pi]} |g(x-x_0)| < \infty
\end{aligned}$$

The uniqueness follows with the same lemma as before. □

If we now compute the STFT for a fixed x_0, we get the Fourier representation to describe the function at x_0. The choice of the window function g has a great impact on the accuracy of the analysis. A wider window would average over all frequencies inside the window, distorting the result. But a small window (the δ-distribution as an extreme) will not always yield the best results either. The uncertainty principle is a natural border for the accuracy. As its written in [5], the following inequation holds.

Theorem 2.1.8. *Let f and g be in $C^\infty[0,2\pi] \subset S([0,2\pi])$, $x_0 \in [0,2\pi]$ and $\xi_0 \in \mathbb{R}$. With $u(x) = f(x)g(x-x_0)$ follows*

$$\left(\int_0^{2\pi} (x-x_0)^2|u(x)|^2dx\right)^{\frac{1}{2}} \cdot \left(\int_{-\infty}^{\infty} (\xi-\xi_0)^2|\hat{u}(\xi)|^2d\xi\right)^{\frac{1}{2}} \geq \frac{1}{2}||fg||_{L_2}^2 \quad (2.9)$$

Proof. A proof can be found in ([5], pp. 21-24) □

This border helps us differentiate between window functions. A good window function would have a very small difference between the two sides, possibly 0. Luckily, this window function exists.

Theorem 2.1.9. *Let g be the gaussian core*

$$g(x) = \frac{1}{2\sqrt{\pi\sigma}} e^{-\frac{1}{4\sigma}(x)^2} \quad (2.10)$$

Then the previous inequation holds.

Proof. The gaussian core is an eigenfunction of the Fourier transformation, so $u(x) = \hat{u}(x)$. Together with $||g||_{L_2} = 1$ the rest follows straight forward. □

Because of this property, the STFT with a gaussian core as its window function has its own name.

2.1.3 The Gabor-Transformation

Theorem 2.1.10. *Let f be in $C^\infty[0, 2\pi] \subset S([0, 2\pi])$. Then*

$$c_{k,x_0} = \frac{1}{2\pi} \int_0^T \frac{1}{2\sqrt{\pi\sigma}} e^{-\frac{1}{4\sigma}(x-x_0)^2} f(x) e^{-ikx} dx, k \in \mathbb{Z} \tag{2.11}$$

are called the Gabor coefficients of f centered around x_0.

σ can be varied to control the width of the window to either get a good resolution in time ($\sigma \to 0$) or frequency ($\sigma \to \infty$), resulting in a bad resolution in the other area, but the multiplication of both resolutions are optimal for any σ. The transformation is named after Dennis Gabor, who 1947 took Fouriers work to adapt it to changing frequencies([8]). These gabor coefficients will now be used to actually implement an analysis. This implementation will always result in a transfer from analytic to discrete computations, so a few things have to be clarified.

2.2 Discrete Preconsiderations

2.2.1 Discrete Frequencies

Sound waves are continous functions, but the digitalisation does not allow anything continous. In fact, a *sample and hold step* is used, saving the value of the incoming signal and storing it until the next step occurs(cf.[6]). This completely neglects the difference between the steps, so an inaccuracy is created, quantification noise ([6]).

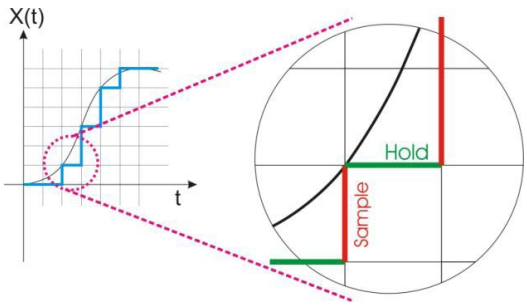


Figure 2.2: A sample and hold step

Smaller steps will result in a more precise reconstruction of the incoming signal. Big steps need less data space but can distort the result by analysing

frequencies that are not there. This effect is known as under sampling. The red dots symbolize an undersampling, recreating a function with a lower frequency that has nothing to do with the real frequency.

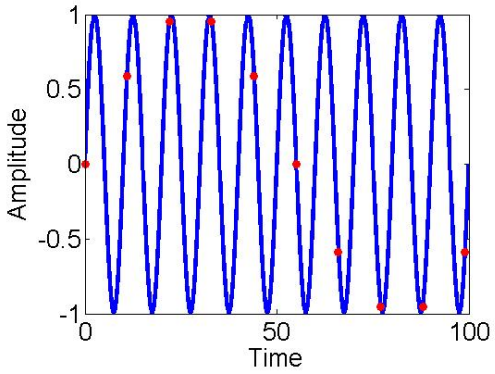


Figure 2.3: Undersampling

A good compromise can be found with the following theorem, as its written in [6]. A proof can be found in ([2], pp.120-121).

Theorem 2.2.1. *A signal will be sampled exactly, if the sampling frequency is at least the double of the highest frequency that shall be analysed.*

$$f_s > 2f_{max} \tag{2.12}$$

As mentioned before, the highest audible frequency is 20 kHz. A standard sampling frequency is 44100 Hz, satisfying the theorem. And because 44100 is equal to $2^2 3^2 5^2 7^2$, each second can be divided by many different natural numbers and still have natural factors.

2.2.2 Noise

Noise is an "unwanted residual electronic noise signal that gives rise to acoustic noise"[7]. When analysing anything real, noise is always an issue, because microphones and sound systems produce distortions in the music. A theoretical perfect tone of i.e. 10 kHz will result in a broad spectrum of different frequencies, only peaking at 10 kHz in the Fourier space. This can easily be taken care of by using thresholds as long as the played music is louder than any background noise.
With all continous and discrete effects considered we can now begin the implementation.

Chapter 3

Numerical implementation

3.1 Idea

Teaching a computer to distinguish between a piano and a violin is impossible without having something to differentiate between them. There are several possibilities to solve this problem. One could analyse each instrument to get its characteristics of overtones and then store this information to compare an incoming sound with different instrumental characteristics, but this approach fails because of the uncertainty principle. The overtone characteristics can only be computed for the natural multiples but in reality a wide spectrum of frequencies around this multiples exhibit high amplitudes. A better approach is to create a dictionary out of easy to analyse tones with the proper informations about these tones given by a user. The resulting implementation will have two parts. First a dictionary $D = \{\varphi_1 \in Y | i \in \mathbb{N}\}$ is built out of the gabor transformations of analysed tones, described with pitches and corresponding instruments. Then the synthesization $A : l_2 \to Y, x \mapsto \sum x_i \varphi_i$ creates an operator A. The real analysis of a piece of music computes the gabor transformation at each point and then uses an operator to find the best fit. This result is stored and at the end transformed into a partiture.

3.2 Minimization tools

Finding a good operator is the hardest part. The algorithm has to find the best match but also has to be aware of highly correlated accords. There are mainly 3 algorithms that are suited for this case:

3.2.1 The simple L_2 comparison

$$\min_x ||Ax - b||_2^2 \tag{3.1}$$

The minimum can be found by solving $\nabla||Ax - b||_2^2 = 0$. Computing this leads to the gaussian normal equation.

$$\begin{aligned} & 0 = \nabla||Ax - b||_2^2 && (3.2)\\ \Leftrightarrow \quad & 0 = \nabla(Ax - b)^T(Ax - b) && (3.3)\\ \Leftrightarrow \quad & 0 = \nabla x^T A^T Ax - 2(A^T x, b) + ||b||_2 && (3.4)\\ \Leftrightarrow \quad & 0 = A^T Ax - A^T b && (3.5)\\ \Leftrightarrow & A^T b = A^T Ax && (3.6) \end{aligned}$$

There are several algorithms for the compution. See [1] for further details. Each coordinate of x stands for the amplitude of a played tone, so with a threshold to ignore the noise we can tell which $\varphi \in D$ are playing. Sadly the overtones of different pitches overlap, disturbing the result by computing high amplitudes for tones with different base frequencies but similar overtones. A desired property of the operator would be to distinguish between played ($x_i \neq 0$) and unplayed tones ($x_i = 0$), possibly with very few pitches marked as played.

3.2.2 The sparsity approach

This attribute can be achieved by adding a penal term

$$\min_x \frac{1}{2}||Ax - b||_2^2 + \alpha||x||_1 \tag{3.7}$$

to punish any value in $x \neq 0$. If we want to further analyse this effect, we need some preconsiderations, as they can be found in [5]. Due to the non differentiable $||x||_1$ we can not build the gradient of the term that shall be minimised. Instead we will use the subdifferential, which can be defined for convex functions. A definition for convexity can be found in [1].

Definition 3.2.1. *Let $f : l^2 \to R$ be a convex function. Then*

$$\partial f(x_0) = \{g \in \mathbb{R}^n : f(x) - f(x_0) \geq g^T(x - x_0) \forall x \in \mathbb{R}^n\} \tag{3.8}$$

*Each element in $\partial f(x_0)$ is called **subgradient**. If f is diffentiable, the subgradient equals the normal gradient.*

This definition can be used to compute the subgradient of the given algorithm. Solving $0 \in \partial f$ results in the x that minimises (3.7), because we can then examine the subgradient for g=0:

$$f(x) - f(x_0) \geq 0 \Leftrightarrow f(x) \geq f(x_0) \forall x \in \mathbb{R}^n$$

Thus, x_0 is the minimal value of f. The following lemma now computes the subdifferential of the convex function $f(x) = |x|$.

Lemma 3.2.2. *Suppose* $f : l^2 \to R, x \mapsto |x|$ *and* $i \in \{1, ..n\}$. *Then*

$$\partial f(\bar{x})_i = \begin{cases} \{-1\}, & \text{if } \bar{x}_i < 0 \\ [-1, 1], & \text{if } \bar{x}_i = 0 \\ \{1\}, & \text{if } \bar{x}_i > 0 \end{cases} \tag{3.9}$$

This function is called ***Sign(x)****.*

Proof. Suppose $\bar{x} < 0$ and fix i. Then

$$\partial f(\bar{x})_i = \{g_i \in \mathbb{R} : f(x_i) + \bar{x}_i \geq g(x_i - \bar{x}_i) \forall x_i \in \mathbb{R}\} = \begin{cases} [-1, \infty] & \text{if } x_i < 0 \\ [-\infty, -1] & \text{if } x_i > 0 \end{cases} \tag{3.10}$$

$\Rightarrow \partial f(\bar{x})_i = [-1, \infty] \cap [-\infty, -1] = \{-1\}$. The other statements follow in an analogous manner. □

Now we can analyse the subdifferential of (3.7).

$$\partial \left(\frac{1}{2} ||Ax - b||_2^2 + \alpha ||x||_1 \right) \tag{3.11}$$

$$= A^T(Ax - b) + \alpha Sign(x) \tag{3.12}$$

It follows that the optimal x solves

$$-A^T(Ax - b) \in \alpha Sign(x) \tag{3.13}$$

$$\Leftrightarrow x - A^T(Ax - b) \in x + \alpha Sign(x) \tag{3.14}$$

The right side can be visualised by the following picture, showing $\alpha Sign(x)$ for x-values between -2α and 2α. $\alpha Sign(x)$ can easily be geometrically inverted by mirroring it at the bisecting line.

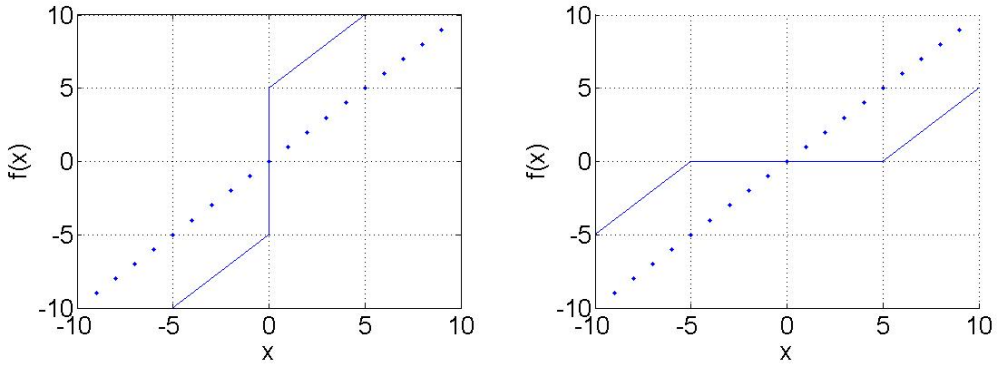


Figure 3.1: $x + \alpha Sign(x)$ (left) and its inverse (right) with $\alpha = 5$

This inverse is unique and well defined. An exact derivation can be found in [1]. If we can now show that a x exists, which fulfills $x - A^T(Ax - b) \in x + \alpha Sign(x)$, we can begin to build a simple fixpoint iteration. For the existence, examine the following theorem, taking out of [2]

Theorem 3.2.3. *Let X be a reflexive Banachspace, $F : X \to R_\infty$ bounded from below, coercive and lower semi-continous. Then there exists a minimizer for F in X.*

Proof. A proof can be found in [2] aswell. □

With this theorem, we can prove the existence for our special case.

Lemma 3.2.4. *$F : l_2 \to R_\infty$, $x \mapsto \frac{1}{2}||Ax-b||_2^2+\alpha||x||_1$ fulfills the conditions of the previous theorem, thus having a minimizer.*

Proof. With l_2 being reflexive and F begin bounded from below by 0 (attribute of every norm), the first two attributes are easy to see.
Coercivity: Because of $||x||_2 < ||x||_1$, it is easy to see, that if $||x||_2$ (the norm of l_2) converges towards ∞, the functional written as $F(x) = ||x||_1 + const$ converges towards ∞ aswell.
The lower semi-continuity follows with the lemma of Fatou [1]. In conclusion, all conditions are fulfilled, guaranteeing the existence of a minimizer. □

If we now want to actually implement this minimization, we can use the inverse S_α for a simple fixpoint iteration.

$$S_\alpha(x - A^T(Ax - b)) = x \tag{3.15}$$

Start with an arbitrary x_k. Compute $x_{k+1} = S_\alpha(x_k - A^T(Ax_k - b))$ until $x_{k+1} = x_k$. This algorithm is simple to implement but very slow. There

are several improvements that can be made, but describing them would go beyond the scope of this paper. Further informations about convergence and attributes can be found in [5].
One important attribute of this approach is the namegiving sparsity.

Definition 3.2.5. *Let x be element of l_2. x is called **sparse**, if only a finite number of elements in x are unequal to 0.*

This attribute can easily be seen with the previous images. An iteration would shrink all values between $-\alpha$ and α to 0, monotonely decreasing the number of elements unequal to 0. The space in which x exists, is l_2. In l_2 all sequences are converging to 0. Thus there exists a $N \in \mathbb{N}$, so that for all $n > N : |x_n| < \alpha$, guaranteeing the sparsity after the first step of the iteration. More exactly, the choice of α determines the sparsity. In fact, examine the following theorem.

Theorem 3.2.6. *x=0 is the unique optimal solution if and only if*

$$A^T b \in \alpha Sign(0) = \alpha[-1,1]^n \Leftrightarrow (A^T b)_i \in [-\alpha, \alpha] \tag{3.16}$$

Proof. " $\Rightarrow$ " Suppose x=0 fulfills $-A^T(Ax-b) \in \alpha Sign(x)$. Then

$$\begin{aligned} &-A^T(A \cdot 0 - b) \in \alpha Sign(0) \\ \Leftrightarrow\ &-A^T(0-b) \in [-\alpha,\alpha]^n \\ \Leftrightarrow\ &A^T b \in \alpha[-1,1]^n \end{aligned}$$

" $\Leftarrow$ " Suppose $(A^T b) \in [-\alpha,\alpha]^n \quad \forall i \in 1..n$. Then x=0 **is** an optimal solution.

$$-A^T(Ax-b) = A^T b \in [-\alpha,\alpha]^n$$

Suppose x and 0 are mimimizer of $\frac{1}{2}||Ax-b||_2^2 + \alpha||x||_1$. Then

$$\begin{aligned} &\frac{1}{2}||0-b||_2^2 = \frac{1}{2}||Ax-b||_2^2 + \alpha||x||_1 = \frac{1}{2}||Ax||_2^2 - \langle Ax, b\rangle + \frac{1}{2}||b||_2^2 + \alpha||x||_1 \\ \Leftrightarrow &0 = \frac{1}{2}||Ax||_2^2 - \langle Ax, b\rangle + \alpha||x||_1 \\ \Leftrightarrow &\frac{1}{2}||Ax||_2^2 + \alpha||x||_1 = \langle Ax, b\rangle = \langle x, \underbrace{A^T b}_{\in[-\alpha,\alpha]^n} \rangle \leq \alpha||x||_1. \end{aligned}$$

It follows that $\frac{1}{2}||Ax||_2^2 = 0 \Rightarrow Ax = 0$. Inserting this result into the previous equation follows $\alpha||x||_1 = 0 \Rightarrow x = 0$. So x=0 is the **unique** solution. □

In conclusion, choosing $\alpha \geq \max_{i\in\{1,..,n\}} |A^T b|_i$ will result in no played tone. But α does not only yield in this defining border. It also determines the norm of the optimal x, because the norm monotonely decreases with increasing α.

Theorem 3.2.7. *Suppose $\alpha_1 > \alpha_2$ and let x_1 and x_2 be the optimal minimizer of the corresponding term. Then*

$$||x_2||_1 \geq ||x_1||_1 \tag{3.17}$$

Proof. If x_1 is the minimizer of $\frac{1}{2}||Ax-b||_2^2 + \alpha_1||x||_1$ and x_2 the minimizer of the analogues term, it follows that

$$\begin{aligned}\frac{1}{2}||Ax_2-b||_2^2 + \alpha_1||x_2||_1 &\geq \frac{1}{2}||Ax_1-b||_2^2 + \alpha_1||x_1||_1\\ \frac{1}{2}||Ax_1-b||_2^2 + \alpha_2||x_1||_1 &\geq \frac{1}{2}||Ax_2-b||_2^2 + \alpha_2||x_2||_1\end{aligned}$$

These terms are equal to

$$\begin{aligned}\frac{1}{2}||Ax_2-b||_2^2 + \alpha_1(||x_2||_1 - ||x_1||_1) &\geq \frac{1}{2}||Ax_1-b||_2^2\\ \frac{1}{2}||Ax_1-b||_2^2 &\geq \frac{1}{2}||Ax_2-b||_2^2 + \alpha_2(||x_2||_1 - ||x_1||_1)\end{aligned}$$

combined to

$$\begin{aligned}\frac{1}{2}||Ax_2-b||_2^2 + \alpha_1(||x_2||_1 - ||x_1||_1) &\geq \frac{1}{2}||Ax_2-b||_2^2 + \alpha_2(||x_2||_1 - ||x_1||_1)\\ \Leftrightarrow \alpha_1(||x_2||_1 - ||x_1||_1) &\geq \alpha_2(||x_2||_1 - ||x_1||_1)\\ \Leftrightarrow (\alpha_1-\alpha_2)(||x_2||) &\geq (\alpha_1-\alpha_2)(||x_1||)\end{aligned}$$

For $\alpha_1 > \alpha_2$ the result $||x_2||_1 \geq ||x_1||_1$ follows. □

In conclusion, α is the only threshold we need to determine between pauses and played tones.

3.2.3 The elastic net approach

Highly correlated entries in the dictionary result in numerical problems. The correlation A^TA gets instable when solved against A^Tb. A solution can be achieved by adding a second penal term to seperate highly correlated terms. This leads us to the elastic net approach:

$$\min_x \frac{1}{2}||Ax-b||_2^2+\alpha||x||_1+\frac{1}{2}||\sqrt{\beta}x||_2^2 \tag{3.18}$$

To research existence and uniqueness of minimizers, observe the following lemma.

Lemma 3.2.8.

$$\frac{1}{2}||Ax-b||_2^2+\alpha||x||_1+\frac{1}{2}||\sqrt{\beta}x||_2^2=\frac{1}{2}||\begin{pmatrix}A\\ id\sqrt{\beta}\end{pmatrix}x-\begin{pmatrix}b\\ 0\end{pmatrix}||_2^2+\alpha||x||_1 \tag{3.19}$$

Proof.

$$\begin{aligned}
&\frac{1}{2}||\begin{pmatrix}A\\ id\sqrt{\beta}\end{pmatrix}x-\begin{pmatrix}b\\ 0\end{pmatrix}||_2^2+\alpha||x||_1\\
&=\frac{1}{2}||\begin{pmatrix}Ax-b\\ id\sqrt{\beta}\end{pmatrix}||_2^2+\alpha||x||_1\\
&=\frac{1}{2}\left(\sum_{i=1}^{n_1}(Ax-b)_i^2+\sum_{i=n_1+1}^{n}(\sqrt{\beta}x)^2\right)+\alpha||x||_1\\
&=\frac{1}{2}||Ax-b||_2^2+\alpha||x||_1+\frac{1}{2}||\sqrt{\beta}x||_2^2
\end{aligned}$$

□

In conclusion, the elastic net minimization term can be written as in the sparsity approach, guaranteeing all attributes shown before. The effect of β is to numerically stabilize the algorithm without improving the overall quality of the algorithm. This addition causes the algorithm to get slower, so the following algorithm for the sound analysis will use the sparsity algorithm. For further details and applications of the elastic net approach, see [10].

3.3 Actual implementation

With the described algorithms we can now begin to create the main algorithm for the analysis, divided into small parts. The whole code can be found in the addendum and is distributed on the CD in the back of this thesis.

3.3.1 Building the operator

First, the operator has to be built to analyse the incoming sound files. When reading in sound files, the data is stored in its oscillating wave form. The first step is to set the frequencies that shall be analysed. Each frequency will be compared inside the operator, so that a bigger array of frequencies will result in a better approximation, but is mainly responsible for the data storage and time needed. A good compromise is to analyse all frequencies associated with and exactly between the keys of a piano.

Algorithm 1 Frequencies

$F(1) \leftarrow 27.5$
$root \leftarrow \sqrt[24]{2}$
for $i = 2 \rightarrow 152$ **do**
  $F(i) \leftarrow F(i-1) * root$
end for

The keys of a piano are build in a logarithmic scale, that can be recreated by this algorithm. Using $\sqrt[24]{2}$ instead of $\sqrt[12]{2}$ additionally stores all frequencies between the keys. With frequencies clarified, we can know start filling the operator with fitting sound files. There are strong specifications towards eligible files. They should consist of unisonous clearly distinguished and increasing pitches with as little noise as possible. Such a sound file can be used to seperate and store its tones. The seperation is shown by the following algorithm.

Algorithm 2 SeperateTones(file)

$x \leftarrow wavread(file)$
$threshold \leftarrow max(x)/5$
for $i = 1 \rightarrow size(x, 1)$ **do**
  $ampl \leftarrow max(x(i - 999 : i))$
  if $ampl > threshold, lock == false$ **then**
    $beat(index) \leftarrow i$
    $index \leftarrow index + 1$
    $lock \leftarrow true$
  end if
  if $ampl < threshold/2, lock == true$ **then**
    $beat(index) \leftarrow i$
    $index \leftarrow index + 1$
    $lock \leftarrow false$
  end if
end for

This algorithm stores the start and end of each pitch in an array called beat, so that the following storing algorithm can assign the right pitch to the corresponding spectrogram.

Algorithm 3 StoreTones(file,tonestart,toneend)

$x \leftarrow wavread(file)$
$F \leftarrow Frequencies()$
$beat \leftarrow seperateTones(file)$
$toneheight \leftarrow tonestart : toneend$
for $i = 1 : 2 : size(beat, 1) - 1$ **do**
  $wave \leftarrow x(beat(i) : beat(i + 1));$
  $Tone \leftarrow abs(spectrogram(wave, 1024, 140, F, 44100))$
  $Tone \leftarrow Tone/norm(Tone)$
  $Operator(:, index) \leftarrow Tone$
  $height(1, index) \leftarrow toneheight(index)$
  $Instrument(:, index) \leftarrow instrum$
  $index = index + 1$
end for
$save('Dictionary','number','Operator','height','Instrument')$

The algorithm computes the matlab-intern routine ‘spectrogram’ to get the gabor transformation. See the matlab documentation for further information.

The operator is synthesized by storing these results in a matrix A. In addition to that, the height and instrument of the pitches are stored in a second vector. Initially, these informations must be inserted by the user to prepare the computer for the analysis. Without a proper operator the analysis can not work, so its quality is crucial.

3.3.2 Analysing a sound file

With the operator synthesized, we now can use it to analyse an arbitrary sound file. The algorithm can be divided into four steps.

Load the operator

First the operator with the corresponding pitches and instruments has to be loaded. The informations about the tones are later used to create the score of the song.

Create the spectrogram of the sound file

It is important to compute the spectrogram with the exact same attributes as the ones in the dictionary. Only this leads to a proper comparison between the amplitudes of the frequencies.

Analyse each part of the sound file

Line by line the spectrogram can now be compared with the operator using any minimization algorithm. The program presented here uses the sparsity algorithm. The choice of α depends on the character of the sound file. As seen before the term $\alpha = \max_{i\in\{1,..,n\}} |A^T b|_i$ describes the border towards $x = 0$ as the optimal minimum, so $\alpha = \max_{i\in\{1,..,n\}} |A^T b|_i - 10^-10$ will give us exactly one pitch unequal to zero, the played pitch. In practice, $\alpha = 35$ gives satisfying results aswell and also gets two or three pitches at a time (chords).

Output the results

The algorithm has two visual outputs. The first one describes the length and heights of the playing instruments at any time by plotting the heights of the minimized x that are unequal to zero against time. The x axis tracks the time. On the y axis the frequencies are shown, encrypted in the heights of the tones we normally hear. The transformation between y and its corresponding frequency is $f = 27.5\ \sqrt[12]{2^y}$. This visualisation helps

identifying the length and heights of tones, but is unfamiliar and unpractical to the performing musician. The second visual output is the real score which is created out of the corresponding symbols for the lengths and play times of the instruments, showing the same information in another manner.

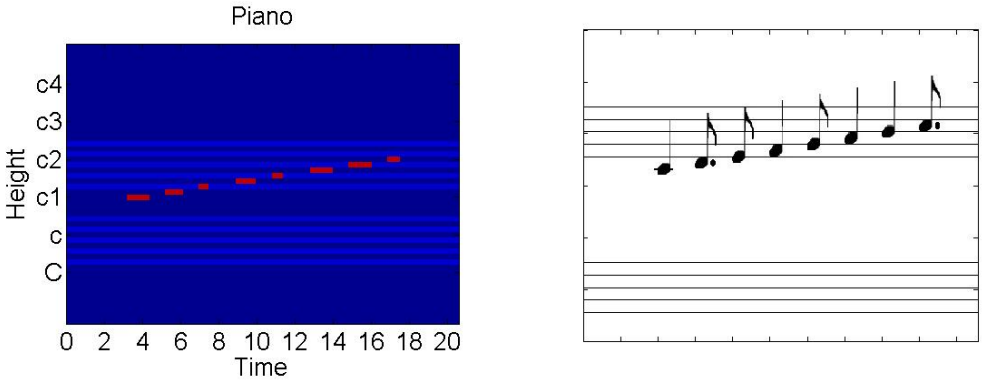


Figure 3.2: Different visualizations

All these programs can be combined into one GUI, shown in the following picture.

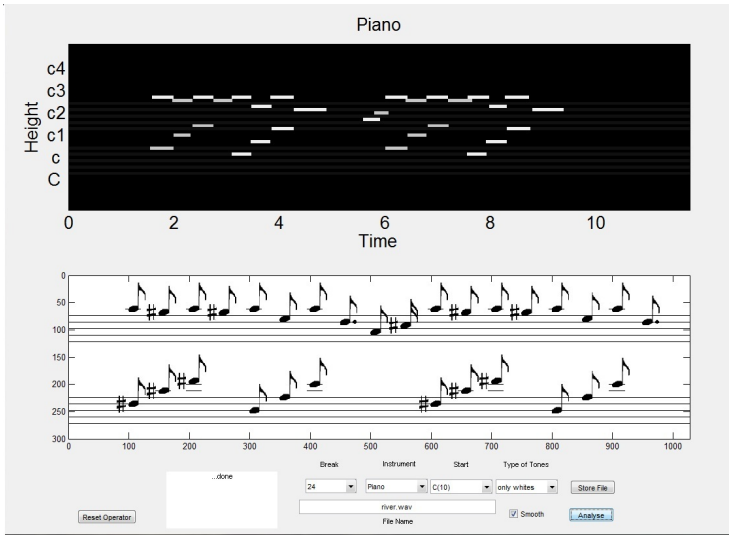


Figure 3.3: GUI

3.4 Results and Plausability

Analysing results of the algorithm has to cover all kinds and complexities of different music styles. The most simple music has only one instrument, clear pitches, that are seperated from each other and no chords. This constraint seems very strict, but there are many examples to easily test the algorithms. Nearly all nursery rhymes fulfill this attribute and the inability of the human voice to sing more than one pitch at a time results in many well known melodies that can be extracted out of more complex pieces of music.
The probably best known example is the song 'Happy birthday' by Mildred J. Hill. Despite its simple melody, the wavfile is sufficiently complex. With a sharp eye one can see different tones, but it is impossible to tell which frequencies or pitches are present. The spectrogram built with the gabor transformation helps us identify the changing frequencies. In respect to the wav file, the spectrogram is mirrored at the bisecting line. The frequencies are present on the x-axis and increase from left to right. The amplitudes are shown by color. The range starts by blue (no amplitude) and ends by red (high amplitude).

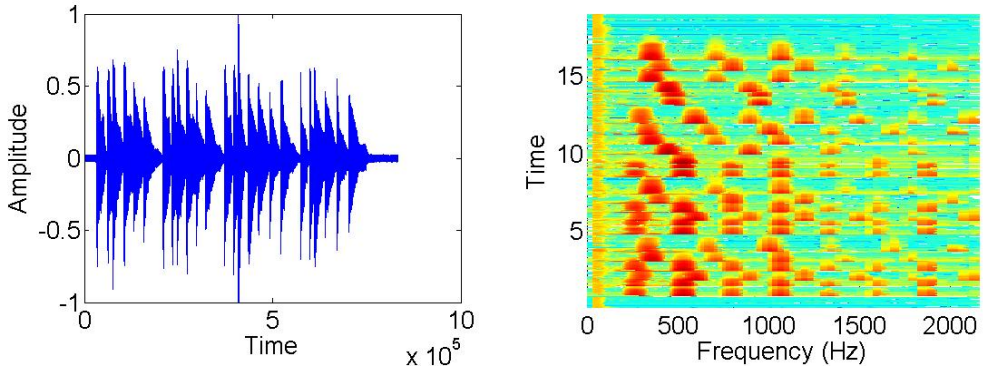


Figure 3.4: Happy Birthday: wavfile (left), spectrogram (right)

The operator is filled with each key of a standard keyboard, resulting in 61 tones from C to c^4. In this range, every row of the operator has an corresponding entry in a vector, that keeps track of the height and instrument of the saved tone, so any arbitrary tone in this range can be analysed. Using this operator the analysis of 'Happy Birthday' results in:

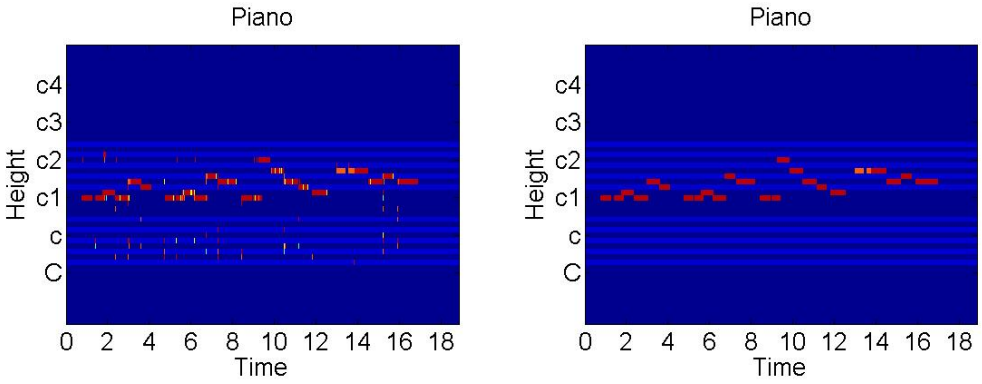


Figure 3.5: Different visualizations

The left picture shows the result before the median filter. The main melody can be seen, but there are several short noise-like tones. As mentioned before, the uncertainty principle forbids the precise analyse of such short tones, so it is safe to say that these are noise related errors. The right picture shows the same result after a filter, that deletes small transients and unifies the reliability of pitches. The main melody has not been affected at all, but all noise effects are gone. This second picture can now be used to recreate the real score.

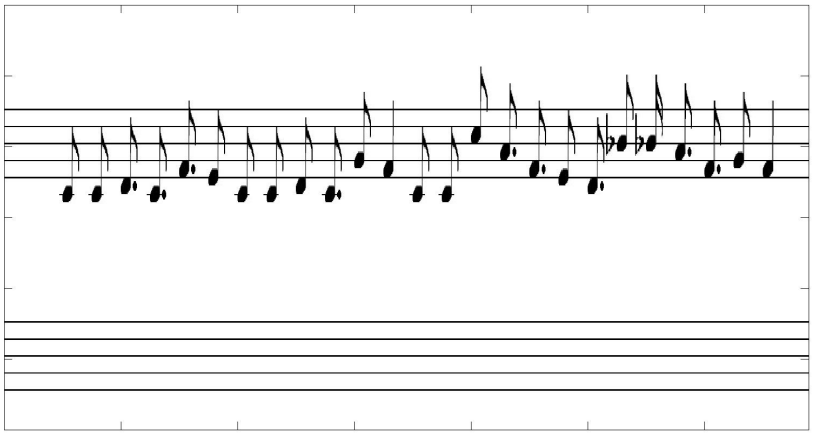


Figure 3.6: Recreated score

This score shows exactly the real score, that was used to record the song.

The lengths of the notes should all be eigths, but this sound file was created on a real instrument, resulting in slightly off lengths. This effect is desired to extract informations about melodic attributes like swing feeling that can not be seen in a normal score.
This song was very easy to analyse. Because of its type its safe to say, that as long as there is only one tone at a time, it can be recognized when the corresponding tone is stored in the operator. The next step is the analysis of a piece of music with a melody and a base line. A well known example for a two line piano song is 'A river flows in you' by Yiruma. In an unisonous song it is possible to see in the wav file where tones start and end without having anything to say about the frequencies. But in a simple two line song this is completely impossible. Songs are created in a manner, that at no time no tone is played, so seperating the tones by thresholding is not possible. The spectrogram helps us a lot in identifying frequencies, but the overlap of two tones with their overtones creates a mass of present frequencies, that needs to be untangled to access the corresponding pitches underneath. Luckily this overlap of frequencies is undisturbed, so the sparsity algorithm can compute the factors belonging to the right pitches.

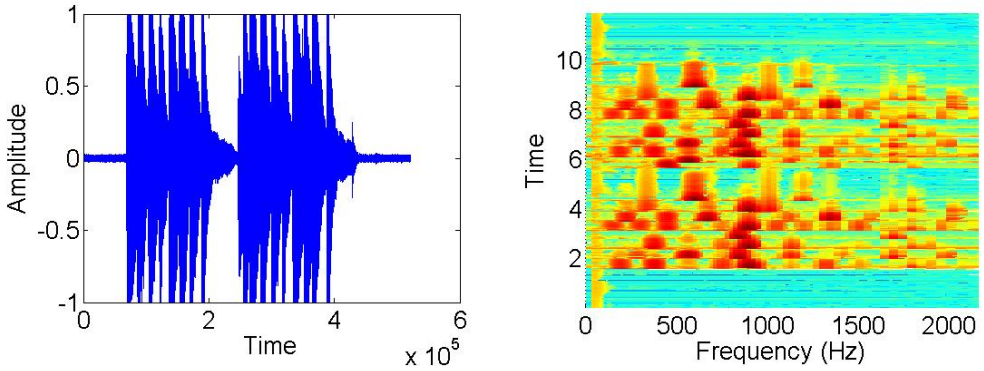


Figure 3.7: A river flows in you: wavfile (left), spectrogram (right)

The operator is still filled with the same tones of a keyboard as before. Because both lines are played on the same instrument, no other tones are needed. The left picture shows the identified tones before the filter. When all noise related effects are filtered, the right picture is created. This result seems wrong, but a skilled eye can see the exact base line (the three lowest tones) and the melody (the upper tones). The red rectangles symbolize tones without sign while the orange rectangles have a cross before them.

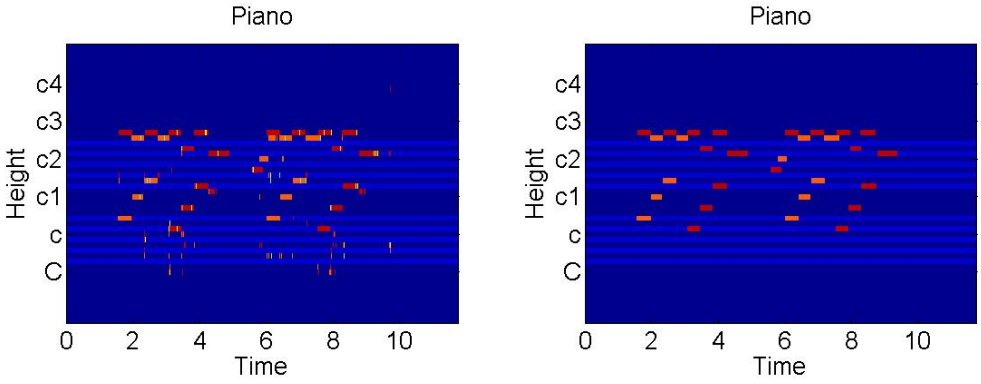


Figure 3.8: Different visualizations

In the score, this can be seen even better. In order to see the base line in the lower stave, the breaking tone to divide between the two staves is set to **e1**. The result shows the exact score, that was used to record it.

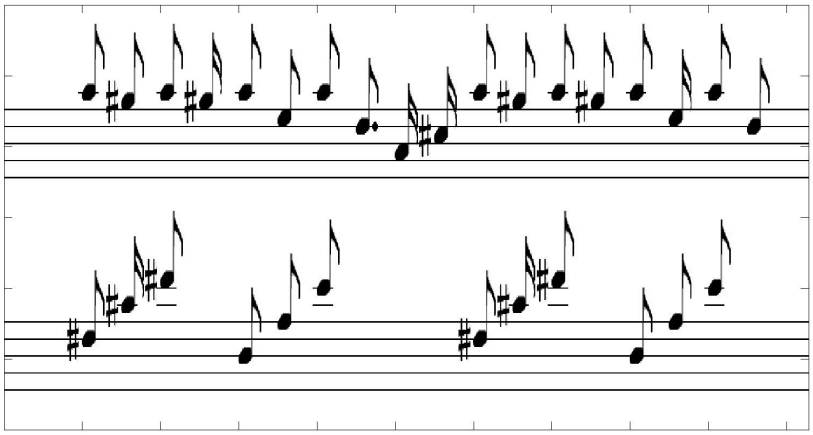


Figure 3.9: Recreated score

Now we can begin to analyse more complex songs. Complexity can increase in many ways. Adding more tones in the spectrum does not affect the analysis at all. In fact, the operator can distinguish even better between different pitches, because the dictionary is filled with more information about the instrument. A good approach to test the quality of the algorithm is to migrate to songs with more than one instrument. The example 'Happy Birthday' is now played simultaneously by a saxophon and a guitar. The guitar is playing one third above the saxophon to distinguish the tones.

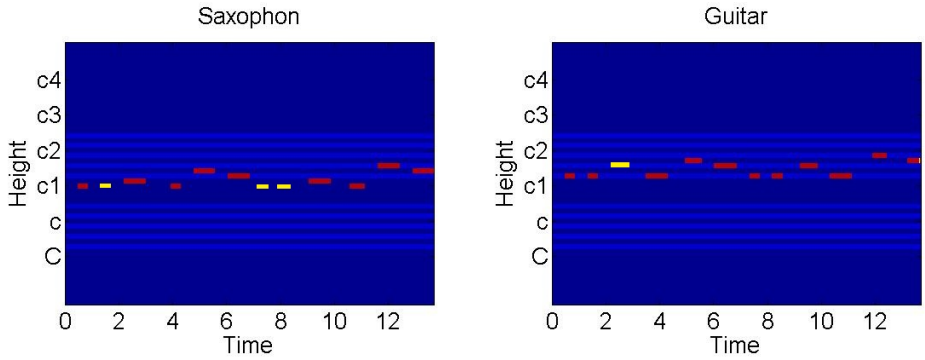


Figure 3.10: Different visualizations: Saxophone (left), Guitar (right)

The analysis shows the melodies nearly perfect. Red pitches stand for a well analysed tone and yellow pitches (artificially inserted after the analysis) symbolize a played tone, that the algorithm did not catch. The quota 20 out of 24 is acceptable for this display. However, this mistakes have a greater impact on the score, because the holes are filled automatically, resulting in:

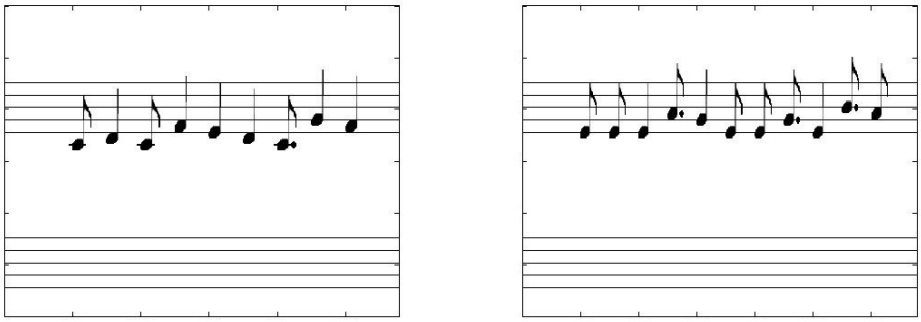


Figure 3.11: Recreated scores: Saxophone (left), Guitar (right)

The melody can barely be seen anymore, so in more complex songs, the score can not be created that easily. Extending the songs to include more instruments increases the number of mistakes made. With these results we can begin to address the limits of the algorithm.

3.5 Limits of the implementation

As long as there is only one instrument, the algorithm works very well. Different instruments will sometimes interfere, because tones have measurable amplitudes long after they can be heard by the human ear. This disturbes the spectrogram, so that the analysis will result in not fitting tones. Between instruments, there is a huge difference in the spectrograms. In the following picture, four instruments are shown, playing the same scale.

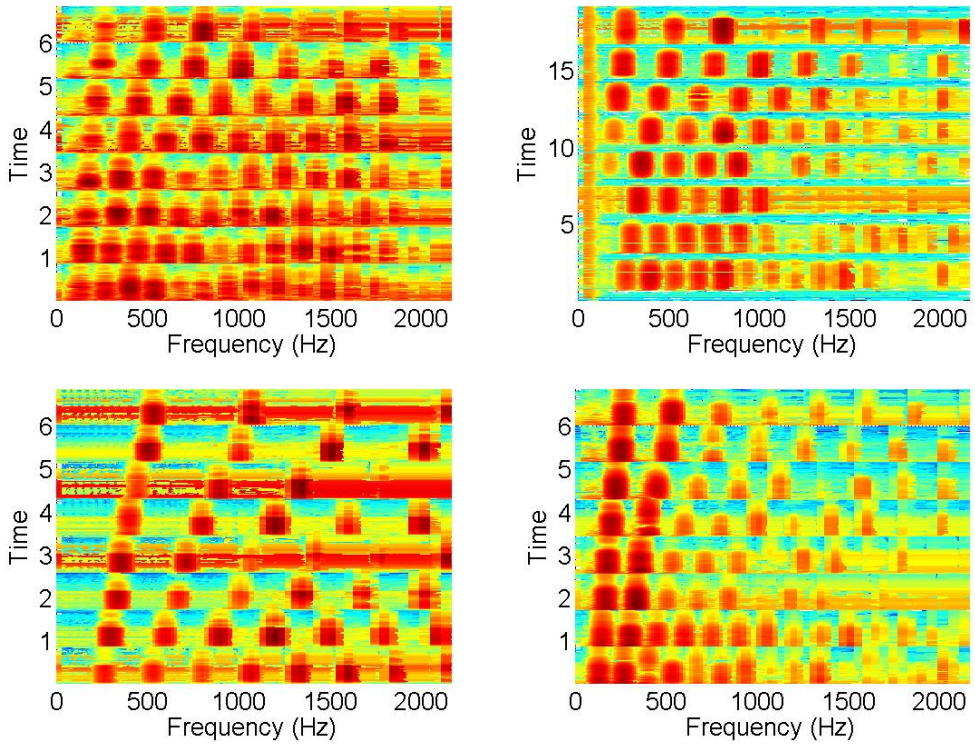


Figure 3.12: Different spectrograms (Guitar, Viola, Saxophone, Piano)

In the upperleft corner, an electric guitar is shown, distorted by many frequencies to create an electronic sound. The upperright picture displays a

viola. All pitches are clearly divided showing all overtones. It is noticable, that the first overtone has a higher amplitude than the base tone itself, complicating the analysis. The lower left display is the spectrogram of a saxophon. Every second overtone is not present, simplyfying the differentiation between two pitches. The forth picture shows a piano, demonstrating the difference of one pitch between four instruments.

Chapter 4

Conclusion and outlook

The results presented show the possibility of analysing frequencies and recreating a score for simple songs. When speaking of more complex songs that are built from several instruments with additional effects to distort the tones, it is hard to say if such a song is possible to analyse in this way. Because of the needed dictionary, each instrument had to be stored once, before the song could get analysed. The complexity and number of different instruments would then result in a database that needs enormous space on the hard drive and took very long to compute. A possible solution would be to analyse the overtone structures of different instruments and find a synthetic way to create a database. The spectrograms of the tones which shall be analysed would then have to be processed to fit the synthetic ones. On this way, the characteristics of different musicians playing the same tone would be ignored, simplifying the analysis. Another mathematical approach would be to use wavelets instead of the STFT to improve the resolution of time to frequency preciseness.
The algorithm described in this paper can only be used to analyse instruments, which play lasting tones of the standard pitches on a piano. However, there are some other important instruments, whose tones can not be allocated to simple pitches. The first one that comes to mind is the standard drum, present in nearly every song. Its tones can more be described as snaps, lasting only for a few milliseconds. But this is not the only difficult insturment. In the wide field of electronic music, there are many other effects that change pitches over time or are so heavily distorted that it is a real challenge to analyse in this way.
In conclusion, there are many parts of sound analysis that are not completely reasearched. But solving the mentioned problems is the work of future papers on this topic.

Appendix A

Code

A.1 Set Frequencies

```
function [ F ] = Frequencies(  )
F=zeros(152,1);
F(1)=27.5;
root=nthroot(2,24);
for i=2:152
    F(i)=F(i-1)*root;
end
```

A.2 Seperate the tones

```
function beat = seperatetones( filename )
%Seperate the tones of a wav-file by using thresholds.
x=wavread(filename); %Load the file
x=x(:,1);
beat=zeros(1000);
index=1;
lock=false;
threshold=max(abs(x))/5; %Set the threshold
ampl=max(abs(x(1:999)));
for i=1001:size(x,1)
    if ampl==abs(x(i-1000))
            ampl=max(abs(x(i-999:i))); %Get the amplitude in each point
    else
            ampl=max(ampl,abs(x(i)));
    end
    if (ampl>threshold && lock==false)||(ampl<threshold/2 && lock==true)
     %Get the start and end of each tone
        beat(index)=i;
        index=index+1;
        lock=~lock;
```

```
    end
end
beat=beat(1:index-1)';
end
```

A.3 Build the operator

```
function StoreData(filename,pitches,start,instrum)
%Stores incoming data in the database
%filename: the filename of the wav-file that shall be stored
%pitches: "both" or "white": the wav-file must provide a monotonic scale.
%Set "both" for a chromatic scale and "white" for white keys.
%start: The starting key. 24 is a C1 (24th key on a piano)
%instrum: The played instrument in range of 1-4. Only important for
%different instruments at a time.
n=200;
F=Frequencies();
x = wavread(filename);
x=x(:,1);
disp('Aquiring Data...');
beat=seperatetones(filename); %Seperate the tones in the file
if  ~exist('Database.mat')
    number=0;
    A=zeros(size(F,1),1000); %Create database
    height=zeros(10,1000);
    Instrument=zeros(1,1000);
    save('Database','number','A','height','waves');
else
    info=load('Database');
    A=info.A;
    height=info.height;
    Instrument=info.Instrument;
end
if strcmp(pitches,'white')==true %Set heights
    tonheight=start:52;
end
if strcmp(pitches,'both')==true
    realindex=1;
    tonheight(1:3)=1:0.5:2;
    for i=start:0.5:52
        index=(i-1)*2+1;
        if(mod(index,14)~=4 && mod(index,14)~=10)
           tonheight(realindex)=i;
           realindex=realindex+1;
        end
    end
end
```

```
tonheight=tonheight';
realindex=1;
disp(['Got ' num2str(size(beat,1)/2) ' tones. Analysing...']);
for index=1:2:size(beat,1)-1 %Divide the wavfile
    if tonheight((index-1)/2+1)<=52
        wave=x(beat(index):beat(index+1));
        Music=spectrogram(wave,1024,140,F,44100); %Analyse the spectrogram
        step=ceil(max(size(Music,2)/n,1));%Set the amount of data extracted
        for j=1:step:size(Music,2)
            tone=abs(Music(:,j));
            tone=tone/norm(tone);
            A(:,realindex+info.number)=tone; %Store the data
            height(1,realindex+info.number)=tonheight((index-1)/2+1);
            Instrument(:,realindex+info.number)=instrum;
            realindex=realindex+1;
        end
    end
    disp([num2str(floor((index+1)/size(beat,1)*100)) '%']);
end
number=info.number+realindex-1;
A=A(:,1:number);
height=height(:,1:number);
Instrument=Instrument(:,1:number);
save('Database','number','A','height','Instrument');
disp('Finished');
```

A.4 Analyse a file

```
function [ Result ] = Analyse( filename, smooth )
% Result=Analyse(filename,smooth)
% Analyses a sound file and shows the partitur.
% Only works when a Database is present. See "StoreData" for more details.
% filename : the filename of the wav-file that shall be analysed
% smooth : setting this on true activates a median filter on the result
wave=wavread(filename);
wave=wave(:,1);
if  exist('Database.mat')
    Database=load('Database'); %Load in the data
else
    error('No database present!');
end
F=Frequencies();
A=Database.A;
height=Database.height;
Instrument=Database.Instrument;
disp('Aquiring Data...');
Music=spectrogram(wave,1024,140,F,44100); %Build the gabor-transformation
disp('...done');
```

```
Result=zeros(52,size(Music,2),floor(max(max(Instrument))));
for k=1:floor(max(max(Instrument)))
    for i=1:5
        for j=1:size(Result,2)
            Result(24+2*i,j,k)=5; %Create the stave of the partitur
            Result(10+2*i,j,k)=5;
        end
    end
end
disp('Analysing pitches...');
for index=1:size(Music,2)
    Note=abs(Music(:,index)); %Analyse each row of the spectrogram
    B=A;
    row=1:size(A,2);
    if size(B,2)>0
       %maxalpha=max(max(abs(B'*Note))-10e-10,10); Activate this for
       %unisonous music
       maxalpha=35;
       rfssResult=rfss(B,Note,maxalpha,0); %Compute the sparsity-algorithm
       rfssResult=abs(rfssResult);
       for i=1:size(rfssResult,1) %Store the pitches with amplitude >0,
                                  %seperate between "sure" and "not sure"
            if  rfssResult(i)>2 && height(row(i))~=0
              Result(floor(height(row(i))),index,floor(Instrument(row(i))))
              =40-20*(height(row(i))-floor(height(row(i))));
            end %Prefixes decrease the value by 10
            if  rfssResult(i)>4 && height(row(i))~=0
              Result(floor(height(row(i))),index,floor(Instrument(row(i))))
              =60-20*(height(row(i))-floor(height(row(i))));
            end

       end
    end
    if floor(index/size(Music,2)*20) ~= floor((index-1)/size(Music,2)*20)
        disp([num2str(floor(index/size(Music,2)*100)) '%']);
    end
end
disp('...done');
for k=1:size(Result,3)
    if smooth==true %if smooth is set, further process the picture
        for m=1:3
        n=6;
        for i=1:size(Result,1)
        for j=n+1:size(Result,2)-n
            E=zeros(4,1);
            if Result(i,j,k)>5 || (Result(i,j-1,k)>5 && Result(i,j+1,k)>5)
                before=true;
                after=true;
                for l=j-n:j-1
```

```
                if before==true
                    if Result(i,l,k)<=5
                        before=false;
                    else
                        E(Result(i,l,k)/10-2)=E(Result(i,l,k)/10-2)+1;
                    end
                end
            end
            for l=j+1:j+n
                if after==true
                    if Result(i,l,k)<=5
                        after=false;
                    else
                        E(Result(i,l,k)/10-2)=E(Result(i,l,k)/10-2)+1;
                    end
                end
            end
            if ~(before||after)
                Result(i,j,k)=Result(i,1,k); %delete small tones
            else
                maxE=max(E);
                for l=1:4
                    if E(l)==maxE
                        Result(i,j,k)=l*10+20;
                        %unify the value to "sure" or "not sure"
                    end
                end
            end
        end
    end
    end
    end
end
if max(max(Result(:,:,k)))>5
    if smooth==true
        Showtones(Result(:,:,k),k); %Create the score
    end
    figure(2*k-1);
    hold off
    plot(0);
    image(Result(:,:,k)); %Plot the result
    set(gca,'YDir','normal')  %Set attributes of the plot
    Tick=0:100:size(Result,2);
    Ticklabel=Tick/50;
    YTick=10:7:50;
    YTicklabel={'C','c','c1','c2','c3','c4'};
    switch k
        case 1
            title('Saxophon');
```

```
            case 2
                title('Guitar');
            case 3
                title('Geige');
        end
        set(gca,'xTick',Tick);
        set(gca,'xTickLabel',Ticklabel);
        set(gca,'yTick',YTick);
        set(gca,'yTickLabel',YTicklabel);
        xlabel('Zeit');
        xlim([0 size(Result,2)]);
        ylabel('Height');
        hold on
    end
end
```

A.5 Showtones

```
function picture=Showtones( Result,k)
%Shows the score of a wav-file.
%This method can only be opened by the method "Analyse".
%See its documentation for further information.

breaktone=28; %C
%breaktone=24; %E
quarter=rgb2gray(imread('Quarter.jpg'));
eigth=rgb2gray(imread('Eigth.jpg'));
sixteenth=rgb2gray(imread('Sixteenth.jpg'));
half=rgb2gray(imread('Half.jpg')); %Load in notes
whole=rgb2gray(imread('Whole.jpg'));
b=rgb2gray(imread('b.jpg'));
f=rgb2gray(imread('f.jpg'));
fs=rgb2gray(imread('Fs.jpg'));
fs=fs(:,1:30);
dot=ones(9,9)*255;
dot(5,2:8)=0;
dot(4,2:8)=0;
dot(6,2:8)=0;
dot(3,3:7)=0;
dot(7,3:7)=0;
dot(2,4:6)=0;
dot(8,4:6)=0;
figure(2*k)
picture=ones(300,10000)*255;
index=2;
match=false;
count=0;
for j=1:size(Result,2)
```

```
for i=1:size(Result,1)
    if Result(i,j)>=50 && i<39 && i>10 %if the tone is inside the range
        length=1;
        height=52-i;
        match=true;
        count=0;
        setdot=false;
        while j+length<size(Result,2)&&Result(i,j+length)==Result(i,j)
            length=length+1; %get length of the tone
        end
        if height <= breaktone %divide score in two parts
            sub=0;
            outsub=0;
        else
            sub=78;
            outsub=12;
        end
        if Result(i,j)==50 %if a prefix is set
        if mod(i,7)==0 || mod(i,7)==3 || mod(i,7)==6
        picture(6*height-48+sub:6*height-19+sub,index*50-19:index*50-5)=f;
        %set an f
        else
        height=height-1;
        picture(6*height-55+sub:6*height-26+sub,index*50-19:index*50-5)=b;
        %set an b
        end
        end
        Result(i,j:j+length)=0; %delete the tone
        length=length-1;
        length=round(length/10);
        tone='none';
        switch length %get the corresponding picture
            case 0
                tone=ones(size(quarter))*255;
            case 1
                tone=sixteenth;
            case 2
                tone=eigth;
            case 3
                tone=eigth;
                setdot=true;
            case 4
                tone=quarter;
            case 5
                tone=quarter;
            case 6
                tone=quarter;
                setdot=true;
            case 7
```

```
                tone=quarter;
                setdot=true;
            case 8
                tone=half;
            case 9
                tone=half;
            case 10
                tone=half;
            case 11
                tone=half;
                setdot=true;
            case 12
                tone=half;
                setdot=true;
            case 13
                tone=half;
                setdot=true;
            case 14
                tone=half;
                setdot=true;
            case 15
                tone=whole;
            case 16
                tone=whole;
        end
        if strcmp(tone,'none')==true
            tone=whole;
            setdot=true;
        end
        aux=picture(6*height-82+sub:6*height-41+sub,index*50:index*50+15);
        picture(6*height-82+sub:6*height-29+sub,index*50:index*50+29)=tone;
        %draw the note
        picture(6*height-82+sub:6*height-41+sub,index*50:index*50+15)=aux;
        if setdot==true
        picture(6*height-38+sub:6*height-30+sub,index*50+20:index*50+28)=dot;
        %draw the dot behind the note
        end
        for k=10+outsub:2:16+outsub
            if height<=k
             picture(6*k-34+sub,index*50-5:index*50+20)=0;
             %draw lines towards to note
            end
            if height>=k+18
             picture(6*(k+18)-34+sub,index*50-5:index*50+20)=0;
            end
        end
    end
end
count=min(10,count+1);
```

```
if count==10&&match==true %as long as a tone is found, a chord is built.
                          %After 10 frames a new index is set
    index=index+1;
    match=false;
end
end
for i=1:5
    for j=1:size(picture,2) %the stave of the score is drawn
        picture(212+12*i,j)=0;
        picture(62+12*i,j)=0;
    end
end

imagesc(picture);
colormap gray;
ylim([0 300]);
xlim([0 index*50+28]);
end
```

Bibliography

[1] Martin Hanke Bourgeois: *Grundlagen der Numerischen Mathematik und des Wissenschaftlichen Rechnens*. Vieweg+Teubner, Wiesbaden, dritte Auflage, 2009.

[2] Kristian Bredies, Dirk Lorenz: *Mathematische Bildverarbeitung, Einführung in die Grundlagen und moderne Theorie*, 2011.

[3] Anssi Klapuri and Manuel Davy: *Signal processing methods for music transcription*, Springer. p. 8. 2006.

[4] J J O'Connor and E F Robertson *MacTutor History of Mathematics archive,Jean Baptiste Joseph Fourier*, 1997. Electronically available at http://www-history.mcs.st-andrews.ac.uk/Biographies/Fourier.html

[5] Stefan Schiffler and Dennis Trede *Einführung in die Bildverarbeitung*, Lecturing script, ZeTeM, University of Bremen, 2011.

[6] Prof. Krieger *Grundlagen der Elektrotechnik* 4, Lecturing script, faculity 3, electronics, University of Bremen, 2011.

[7] *The American Heritage Dictionary of the English Language* Fourth Edition, Houghton Mifflin Company, 2000, archived from the original on June 25, 2008, retrieved May 20, 2010. Electronically available at http://wayback.archive.org/web/jsp/Interstitial.jsp?seconds=5

[8] Nobel commitee *Dennis Gabor, a short biography*, Electronically available at http://www.nobel-winners.com/Physics/dennisgabor.html

[9] Peter Neubäcker *Celemony*, 2011. Electronically available at http://www.celemony.com/cms/index.php?id=news&L=1

[10] Hui Zou and Trevor Hastie *Regularization and variable selection via the elastic net*, Paper 2004.

Declaration

I guarantee that I personally wrote this thesis and there are no passages taken out of other papers without proper citation.

Bremen, July 22, 2011

.....

Tobias Maas